CS202: Computer Organization Final Project

开发者说明

学号	姓名	负责工作	贡献百分比
12010336	黄慧惠	CPU部分（Ifetch，Dmemory），测试场景一asm文件，报告	33.33%
12012719	李思锐	CPU部分（ALU，Led，Switch，MemOrIo，Top），测试场景二asm文件	33.33%
12012902	肖煜玮	CPU部分（Decoder，Controller），Uart，七段数码管显示，矩阵键盘输入，测试场景二asm文件	33.34%

CPU架构设计说明

CPU特性

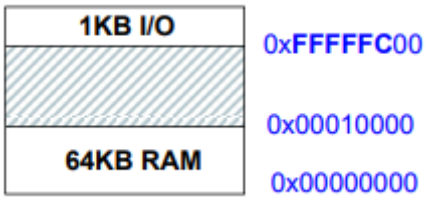
ISA

MIPS指令集

寻址空间设计

冯.诺依曼结构

指令空间、数据空间



外设IO寻址范围

Range	LED(1~16)	LED(17~24)	Switch(1~16)	Switch(17~24)
Address	0xFFFFFC60	0xFFFFFC62	0xFFFFFC70	0xFFFFFC72

矩阵键盘：0xFFFFFC83

CPU的CPI：单周期

CPU接口

UART 信号

```
#          UART 信号
set_property IOSTANDARD LVCMOS33 [get_ports rx]
set_property IOSTANDARD LVCMOS33 [get_ports tx]
set_property IOSTANDARD LVCMOS33 [get_ports start_pg]
set_property PACKAGE_PIN Y19 [get_ports rx]
set_property PACKAGE_PIN V18 [get_ports tx]
set_property PACKAGE_PIN P1 [get_ports start_pg]
```

时钟信号 复位信号 发送键

```
#          时钟信号 复位信号 发送键
set_property IOSTANDARD LVCMOS33 [get_ports fpga_clk]
set_property IOSTANDARD LVCMOS33 [get_ports fpga_rst]
set_property PACKAGE_PIN Y18 [get_ports fpga_clk]
set_property PACKAGE_PIN P20 [get_ports fpga_rst]
```

开关控制模块

```
#          开关控制模块
set_property IOSTANDARD LVCMOS33 [get_ports switches[23]]
set_property IOSTANDARD LVCMOS33 [get_ports switches[22]]
set_property IOSTANDARD LVCMOS33 [get_ports switches[21]]
set_property IOSTANDARD LVCMOS33 [get_ports switches[20]]
set_property IOSTANDARD LVCMOS33 [get_ports switches[19]]
set_property IOSTANDARD LVCMOS33 [get_ports switches[18]]
set_property IOSTANDARD LVCMOS33 [get_ports switches[17]]
set_property IOSTANDARD LVCMOS33 [get_ports switches[16]]
set_property IOSTANDARD LVCMOS33 [get_ports switches[15]]
set_property IOSTANDARD LVCMOS33 [get_ports switches[14]]
set_property IOSTANDARD LVCMOS33 [get_ports switches[13]]
set_property IOSTANDARD LVCMOS33 [get_ports switches[12]]
set_property IOSTANDARD LVCMOS33 [get_ports switches[11]]
set_property IOSTANDARD LVCMOS33 [get_ports switches[10]]
set_property IOSTANDARD LVCMOS33 [get_ports switches[9]]
set_property IOSTANDARD LVCMOS33 [get_ports switches[8]]
set_property IOSTANDARD LVCMOS33 [get_ports switches[7]]
set_property IOSTANDARD LVCMOS33 [get_ports switches[6]]
set_property IOSTANDARD LVCMOS33 [get_ports switches[5]]
set_property IOSTANDARD LVCMOS33 [get_ports switches[4]]
set_property IOSTANDARD LVCMOS33 [get_ports switches[3]]
set_property IOSTANDARD LVCMOS33 [get_ports switches[2]]
set_property IOSTANDARD LVCMOS33 [get_ports switches[1]]
set_property IOSTANDARD LVCMOS33 [get_ports switches[0]]
set_property PACKAGE_PIN Y9 [get_ports switches[23]]
set_property PACKAGE_PIN W9 [get_ports switches[22]]
set_property PACKAGE_PIN Y7 [get_ports switches[21]]
set_property PACKAGE_PIN Y8 [get_ports switches[20]]
set_property PACKAGE_PIN AB8 [get_ports switches[19]]
set_property PACKAGE_PIN AA8 [get_ports switches[18]]
set_property PACKAGE_PIN V8 [get_ports switches[17]]
set_property PACKAGE_PIN V9 [get_ports switches[16]]
```

```
set_property PACKAGE_PIN AB6 [get_ports switches[15]]
set_property PACKAGE_PIN AB7 [get_ports switches[14]]
set_property PACKAGE_PIN V7 [get_ports switches[13]]
set_property PACKAGE_PIN AA6 [get_ports switches[12]]
set_property PACKAGE_PIN Y6 [get_ports switches[11]]
set_property PACKAGE_PIN T6 [get_ports switches[10]]
set_property PACKAGE_PIN R6 [get_ports switches[9]]
set_property PACKAGE_PIN V5 [get_ports switches[8]]
set_property PACKAGE_PIN U6 [get_ports switches[7]]
set_property PACKAGE_PIN W5 [get_ports switches[6]]
set_property PACKAGE_PIN W6 [get_ports switches[5]]
set_property PACKAGE_PIN U5 [get_ports switches[4]]
set_property PACKAGE_PIN T5 [get_ports switches[3]]
set_property PACKAGE_PIN T4 [get_ports switches[2]]
set_property PACKAGE_PIN R4 [get_ports switches[1]]
set_property PACKAGE_PIN W4 [get_ports switches[0]]
```

LED显示模块

```
#           LED显示模块
set_property IOSTANDARD LVCMOS33 [get_ports leds[23]]
set_property IOSTANDARD LVCMOS33 [get_ports leds[22]]
set_property IOSTANDARD LVCMOS33 [get_ports leds[21]]
set_property IOSTANDARD LVCMOS33 [get_ports leds[20]]
set_property IOSTANDARD LVCMOS33 [get_ports leds[19]]
set_property IOSTANDARD LVCMOS33 [get_ports leds[18]]
set_property IOSTANDARD LVCMOS33 [get_ports leds[17]]
set_property IOSTANDARD LVCMOS33 [get_ports leds[16]]
set_property IOSTANDARD LVCMOS33 [get_ports leds[15]]
set_property IOSTANDARD LVCMOS33 [get_ports leds[14]]
set_property IOSTANDARD LVCMOS33 [get_ports leds[13]]
set_property IOSTANDARD LVCMOS33 [get_ports leds[12]]
set_property IOSTANDARD LVCMOS33 [get_ports leds[11]]
set_property IOSTANDARD LVCMOS33 [get_ports leds[10]]
set_property IOSTANDARD LVCMOS33 [get_ports leds[9]]
set_property IOSTANDARD LVCMOS33 [get_ports leds[8]]
set_property IOSTANDARD LVCMOS33 [get_ports leds[7]]
set_property IOSTANDARD LVCMOS33 [get_ports leds[6]]
set_property IOSTANDARD LVCMOS33 [get_ports leds[5]]
set_property IOSTANDARD LVCMOS33 [get_ports leds[4]]
set_property IOSTANDARD LVCMOS33 [get_ports leds[3]]
set_property IOSTANDARD LVCMOS33 [get_ports leds[2]]
set_property IOSTANDARD LVCMOS33 [get_ports leds[1]]
set_property IOSTANDARD LVCMOS33 [get_ports leds[0]]
set_property PACKAGE_PIN K17 [get_ports leds[23]]
set_property PACKAGE_PIN L13 [get_ports leds[22]]
set_property PACKAGE_PIN M13 [get_ports leds[21]]
set_property PACKAGE_PIN K14 [get_ports leds[20]]
set_property PACKAGE_PIN K13 [get_ports leds[19]]
set_property PACKAGE_PIN M20 [get_ports leds[18]]
set_property PACKAGE_PIN N20 [get_ports leds[17]]
set_property PACKAGE_PIN N19 [get_ports leds[16]]
set_property PACKAGE_PIN M17 [get_ports leds[15]]
set_property PACKAGE_PIN M16 [get_ports leds[14]]
set_property PACKAGE_PIN M15 [get_ports leds[13]]
set_property PACKAGE_PIN K16 [get_ports leds[12]]
set_property PACKAGE_PIN L16 [get_ports leds[11]]
```

```
set_property PACKAGE_PIN L15 [get_ports leds[10]]
set_property PACKAGE_PIN L14 [get_ports leds[9]]
set_property PACKAGE_PIN J17 [get_ports leds[8]]
set_property PACKAGE_PIN F21 [get_ports leds[7]]
set_property PACKAGE_PIN G22 [get_ports leds[6]]
set_property PACKAGE_PIN G21 [get_ports leds[5]]
set_property PACKAGE_PIN D21 [get_ports leds[4]]
set_property PACKAGE_PIN E21 [get_ports leds[3]]
set_property PACKAGE_PIN D22 [get_ports leds[2]]
set_property PACKAGE_PIN E22 [get_ports leds[1]]
set_property PACKAGE_PIN A21 [get_ports leds[0]]
```

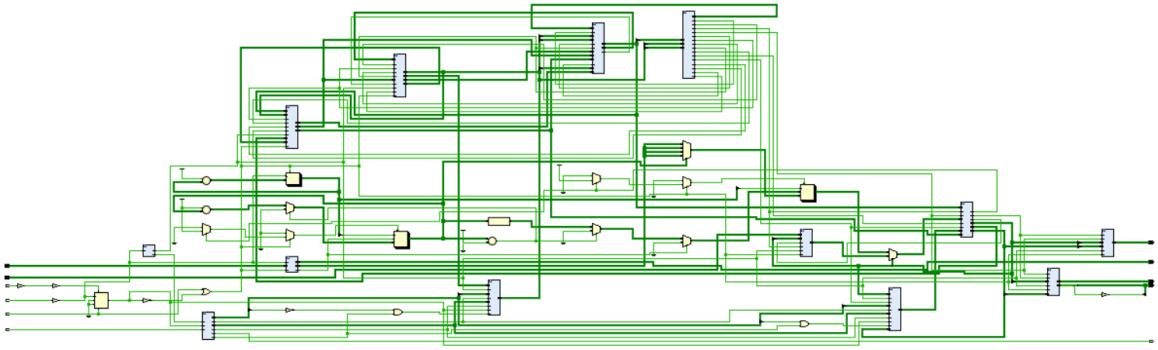
七段数码管显示模块

```
#           七段数码管显示模块
set_property IOSTANDARD LVCMOS33 [get_ports {seg_bit_selection[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_bit_selection[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_bit_selection[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_bit_selection[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_bit_selection[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_bit_selection[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_bit_selection[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_bit_selection[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_selection[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_selection[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_selection[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_selection[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_selection[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_selection[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_selection[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_selection[7]}]
set_property PACKAGE_PIN A18 [get_ports {seg_bit_selection[7]}]
set_property PACKAGE_PIN A20 [get_ports {seg_bit_selection[6]}]
set_property PACKAGE_PIN B20 [get_ports {seg_bit_selection[5]}]
set_property PACKAGE_PIN E18 [get_ports {seg_bit_selection[4]}]
set_property PACKAGE_PIN F18 [get_ports {seg_bit_selection[3]}]
set_property PACKAGE_PIN D19 [get_ports {seg_bit_selection[2]}]
set_property PACKAGE_PIN E19 [get_ports {seg_bit_selection[1]}]
set_property PACKAGE_PIN C19 [get_ports {seg_bit_selection[0]}]
set_property PACKAGE_PIN E13 [get_ports {seg_selection[7]}]
set_property PACKAGE_PIN C15 [get_ports {seg_selection[6]}]
set_property PACKAGE_PIN C14 [get_ports {seg_selection[5]}]
set_property PACKAGE_PIN E17 [get_ports {seg_selection[4]}]
set_property PACKAGE_PIN F16 [get_ports {seg_selection[3]}]
set_property PACKAGE_PIN F14 [get_ports {seg_selection[2]}]
set_property PACKAGE_PIN F13 [get_ports {seg_selection[1]}]
set_property PACKAGE_PIN F15 [get_ports {seg_selection[0]}]
```

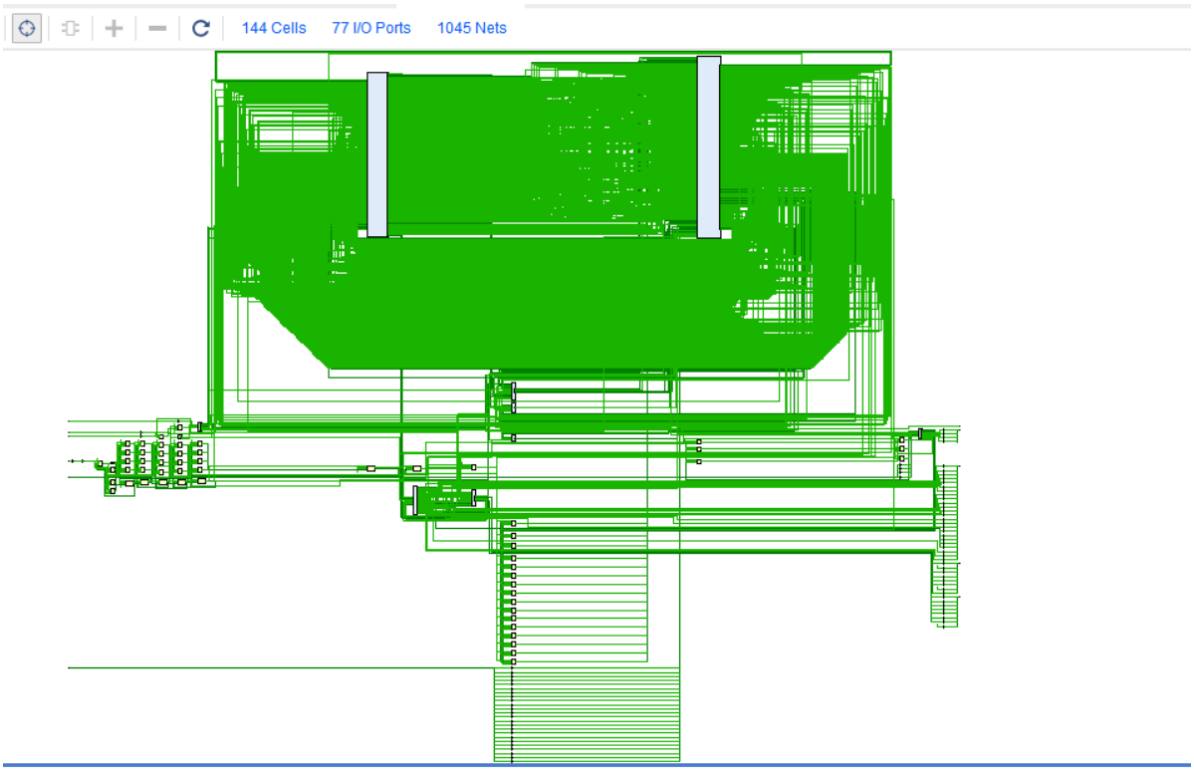
CPU内部结构

CPU内部各子模块的接口连接关系图

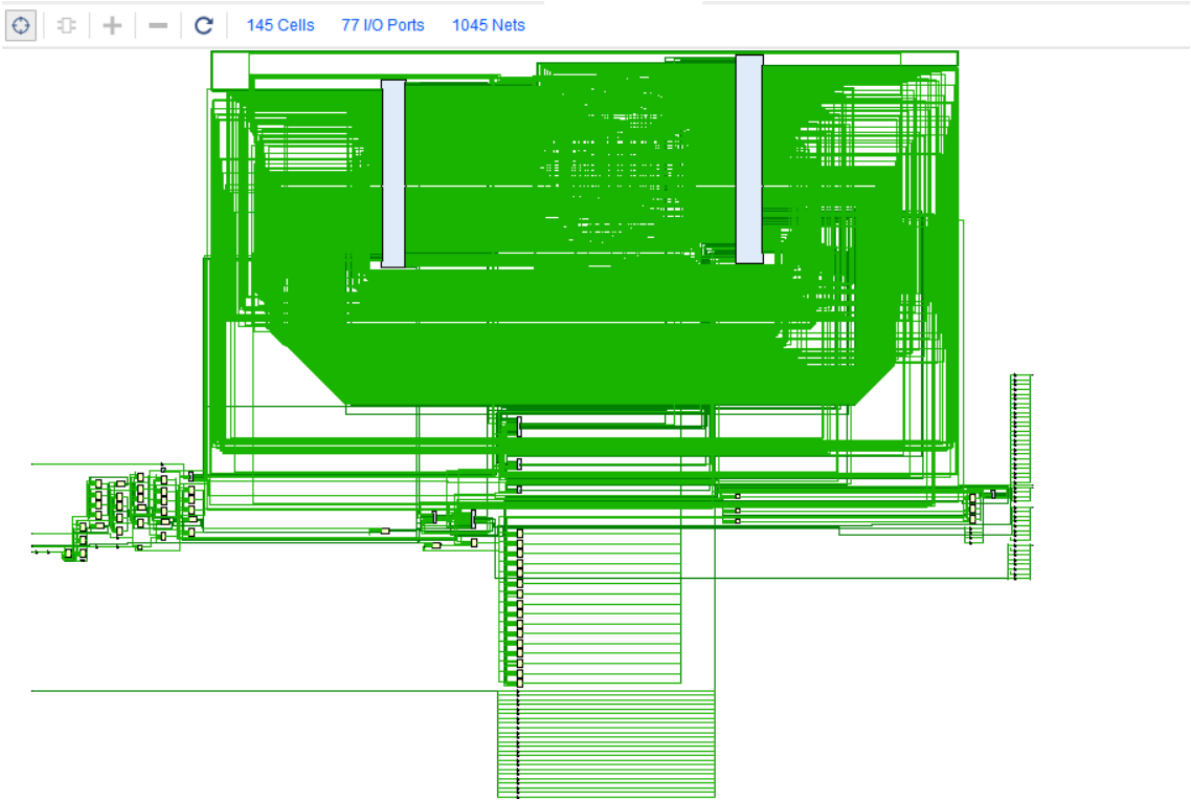
RTL Analysis Schemtic



Implementation Schematic



Synthesis Schematic



CPU内部子模块的设计说明

CPU_top

```
#CPU_top
module CPU_top(fpga_clk, fpga_rst, start_pg, rx, tx, switches, leds,
seg_bit_selection, seg_selection
//, send
// , row, col
//, IORead, stay_PC, Instruction, PC, cpu_fpga_clk
// , ALU_result, address, ioread_data
);
    input fpga_clk, fpga_rst;
//    input send;
    //UART Programmer Pinouts
    // start Uart communicate at high level
    input start_pg;  // Active high
    input rx;       // receive data by UART
    output tx;      // send data by UART
    // UART Programmer Pinouts
    wire upg_clk, upg_clk_o;
    wire upg_wen_o;     // Uart write out enable
    wire upg_done_o;    // Uart rx data have done
    // data to which memory unit of program_rom/dememory32
    wire[14:0] upg_adr_o;
    // data to program_rom or dmemory32
    wire[31:0] upg_dat_o;
    input[23:0] switches;
    output[23:0] leds;
//    output cpu_fpga_clk;
//    output[31:0] Instruction;
    wire cpu_fpga_clk;
    wire[31:0] Instruction;
```

```
    wire[31:0] Read_Data_1;
    wire[31:0] Read_Data_2;
    wire[31:0] mem_data;
    wire[31:0] branch_base_addr;
    wire[31:0] Addr_Result;
    wire[31:0] link_addr;
    wire[31:0] ALU_result;
    wire[31:0] Sign_extend;
    wire[31:0] address;
    wire[31:0] writeData;
    wire[31:0] readData;
    wire[31:0] PC_plus_4;
    wire[15:0] switchrdata;
    wire Jr, RegDST, ALUSrc, MemtoReg, RegWrite, MemWrite, Branch, nBranch, Jmp,
Jal, I_format, Sftmd, Zero;
    wire[1:0] ALUOp;
    wire MemRead, IOWrite, LEDCtrl, SwitchCtrl;
    wire IORead;
    wire[15:0] ioread_data;
    output [7:0] seg_bit_selection;
    output [7:0] seg_selection;
    wire spg_bufg;
    BUFG U1(.I(start_pg), .O(spg_bufg)); // de-twitter
    // Generate UART Programmer reset signal
    reg upg_rst = 1;
    always @ (posedge fpga_clk) begin
        if (spg_bufg) upg_rst = 0;
        if (fpga_rst) upg_rst = 1;
    end
    //used for other modules which don't relate to UART
    wire rst;
    assign rst = fpga_rst | (!upg_rst);
```

Controller

```
#Controller
module control32(
    input[5:0] Opcode, // instruction[31..26], opcode
    input[5:0] Function_opcode, // instructions[5..0], funct
    input [21:0] ALU_resultHigh, // From ALU , indicate data memmory or IO
    output Jr, // 1 indicates the instruction is "jr", otherwise it's not "jr"
    output Jmp, // 1 indicate the instruction is "j", otherwise it's not
    output Jal, // 1 indicate the instruction is "jal", otherwise it's not
    output Branch, // 1 indicate the instruction is "beq" , otherwise it's not
    output nBranch, // 1 indicate the instruction is "bne", otherwise it's not
    output RegDST, // 1 indicate destination register is "rd"(R),otherwise it's
"rt"(I)
    output RegWrite, // 1 indicate write register(R,I(lw)), otherwise it's not
    output MemOrIOtoReg, // 1 indicate read data from memory or I/O port and
write it into register
    output MemRead,  // 1 indicate read data from memory
    output MemWrite, // 1 indicate write data memory, otherwise it's not
    output IORead,   // 1 indicate read data from I/O port
    output IOWrite,  // 1 indicate write data to I/O port
    output ALUSrc, // 1 indicate the 2nd data is immidiate (except "beq","bne")
    output Sftmd, // 1 indicate the instruction is shift instruction
```

```
    output I_format, // 1 indicate the instruction is I-type but isn't
"beq","bne","LW" or "SW"
    output[1:0] ALUOp // if the instruction is R-type or I_format, ALUOp is
2'b10;
                  // if the instruction is"beq" or "bne", ALUOp is 2'b01;
                  // if the instruction is"lw" or "sw", ALUOp is 2'b00;

    );
```

Decoder

```
#Decoder
module decode32(read_data_1,read_data_2,Instruction,mem_data,ALU_result,
                Jal,RegWrite,MemtoReg,RegDst,Sign_extend,clock,reset,opcplus4
                );
    output[31:0] read_data_1;
    output[31:0] read_data_2;
    input[31:0]  Instruction;
    input[31:0]  mem_data;
    input[31:0]  ALU_result;
    input        Jal;
    input        RegWrite;
    input        MemtoReg;
    input        RegDst;
    indicate destination register is "rd"(R),otherwise it's "rt"(I)
    output[31:0] Sign_extend;
    input        clock,reset;
    input[31:0]  opcplus4;              32bits
    reg[31:0] Registers[0:31];
    // R-type : rd = rs + rt
    // I-type : rt = rs + (sign-extended) immediate
    // J-type : jal: reg[31] = opcplus4
    wire [15:0]immediate;
    wire [4:0] rs, rt, rd;
    wire [5:0] opcode;
```

ALU

```
#ALU
module ALU(Read_data_1,Read_data_2,Sign_extend,Function_opcode,Exe_opcode,ALUOp,
           Shamt,ALUSrc,I_format,Zero,Jr,Sftmd,ALU_Result,Addr_Result,PC_plus_4
    );
    // from decoder
    input[31:0]  Read_data_1;      // 从译码单元的Read_data_1中来
    input[31:0]  Read_data_2;      // 从译码单元的Read_data_2中来
    input[31:0]  Sign_extend;      // 从译码单元来的扩展后的立即数

    //from IFetch
    input[5:0]   Function_opcode;  // 取指单元来的r-类型指令功能码,r-form
instructions[5:0]
    input[5:0]   Exe_opcode;       // 取指单元来的操作码
    input[4:0]   Shamt;            // 来自取指单元的instruction[10:6]，指定移位次数
    input[31:0]  PC_plus_4;        // 来自取指单元的PC+4

    //from Controller
    input[1:0]   ALUOp;            // 来自控制单元的运算指令控制编码
```

```
    input         ALUSrc;           // 来自控制单元，表明第二个操作数是立即数（beq，bne除外）
    input         I_format;         // 来自控制单元，表明是除beq，bne，LW，SW之外的I-类型指令
    input         Sftmd;            // 来自控制单元的，表明是移位指令
    input         Jr;               // 来自控制单元，表明是JR指令

    //outputs
    output        Zero;             // 为1表明计算值为0
    output reg [31:0] ALU_Result;   // 计算的数据结果
    output[31:0] Addr_Result;       // 计算的地址结果

    //wires
    wire[31:0] Ainput,Binput;     // the ALU calculation result

    wire[5:0]  Exe_code;          // use to generate ALU_ctrl. (I_format==0) ? Function_opcode : {3'b000, Opcode[2:0]}
    wire[2:0]  ALU_ctl;           // the ontrol signals which affact operation in ALU directly

    wire[2:0]  Sftm;              // identify the types of shift instruction, equals to Function_opcode[2:0]
    reg[31:0]  Shift_Result;      // the result of shift operation

    reg[31:0]  ALU_output_mux;    // the result of arithmetic or logic calculation

    wire[32:0] Branch_Addr;       // the calculated address of the instruction, Addr_Result is Branch_Addr[31:0]
```

programrom

```
#programrom
module programrom(
    rom_clk_i, rom_adr_i, Instruction_o, upg_rst_i, upg_clk_i, upg_wen_i, upg_adr_i, upg_dat_i, upg_done_i
    );
    // Program ROM Pinouts
    input rom_clk_i;
    input[13:0] rom_adr_i;
    output[31:0] Instruction_o;

    // UART Programmer Pinouts
    input upg_rst_i;
    input upg_clk_i;
    input upg_wen_i;
    input[13:0] upg_adr_i;
    input[31:0] upg_dat_i;
    input upg_done_i;

    wire no_Uart_mode = upg_rst_i | (~upg_rst_i & upg_done_i);
```

Ifetch

```
#Ifetch
module
Ifetc32(Instruction,branch_base_addr,Addr_result,Read_data_1,Branch,nBranch,Jmp,
Jal,Jr,Zero,clock,reset,link_addr,PC
//,stay_PC
);
    output[31:0] Instruction;
    output[31:0] branch_base_addr;
    input[31:0]  Addr_result;
    input[31:0]  Read_data_1;
    input        Branch;
    input        nBranch;
    input        Jmp;
    input        Jal;
    input        Jr;
    input        Zero;
    input        clock,reset;
    output reg [31:0]  link_addr;
    output reg [31:0] PC;
    reg[31:0] Next_PC;
```

Dmemory

```
#Dmemory
module dmemory32(fpga_clk,memWrite,address,writeData,readData
,upg_rst_i, upg_clk_i, upg_wen_i, upg_adr_i, upg_dat_i, upg_done_i
 );

    input fpga_clk, memWrite;
    input [31:0] address;
    input [31:0] writeData;
    output[31:0] readData;
    //UART Programmer Pinouts
    input upg_rst_i; // UPG reset (Active High)
    input upg_clk_i; // UPG ram_clk_i (10MHz)
    input upg_wen_i; // UPG write enable
    input [13:0] upg_adr_i; // UPG write address
    input [31:0] upg_dat_i; // UPG write data
    input upg_done_i; // 1 if programming is finished
    wire ram_clk = !fpga_clk;
// CPU work on normal mode when no_Uart_mode is 1.
// CPU work on Uart communicate mode when no_Uart_mode is 0.
    wire no_Uart_mode = upg_rst_i | (~upg_rst_i & upg_done_i);
```

MemoryOrIo

```
#MemoryOrIo
module MemOrIO(mRead, mWrite, ioRead, ioWrite,addr_in, addr_out, m_rdata,
io_rdata, r_wdata, r_rdata, write_data, LEDCtrl, SwitchCtrl);
    input mRead; // read memory, from Controller
    input mWrite; // write memory, from Controller
    input ioRead; // read IO, from Controller
    input ioWrite; // write IO, from Controller
```

```
    input[31:0] addr_in; // from alu_result in ALU
    output[31:0] addr_out; // address to Data-Memory

    input[31:0] m_rdata; // data read from Data-Memory
    input[15:0] io_rdata; // data read from IO,16 bits
    output[31:0] r_wdata; // data to Decoder(register file)

    input[31:0] r_rdata; // data read from Decoder(register file)
    output reg[31:0] write_data; // data to memory or I/O
    output LEDCtrl; // LED Chip Select
    output SwitchCtrl; // Switch Chip Selec
```

LED

```
#LED
module LED(cpu_clk, reset, LEDCtrl, LEDWrite, LED_addr, LEDwdata, LEDout);
    input cpu_clk;
    input reset;
    input LEDCtrl;
    input LEDWrite;
    input[1:0] LED_addr;
    input[15:0] LEDwdata;
    output reg [23:0] LEDout;//rst
```

Switch

```
#Switch
module Switch(switclk, switrst, switchread, switchcs,switchaddr, switchrdata,
switch_i);
    input switclk;      //  时钟信号
    input switrst;      //  复位信号
    input switchcs;     //从memorio来的switch片选信号  !!!!!!!!!!!!!!!!
    input[1:0] switchaddr;  //  到switch模块的地址低端  !!!!!!!!!!!!!!
    input switchread;        //  读信号
    output reg [15:0] switchrdata;      //  送到CPU的拨码开关值注意数据总线只有16根
    input [23:0] switch_i;          //  从板上读的24位开关数据
```

Tube

```
#Tube
module Tub(clk, rst, IOWrite, LED_addr, writeData, Y, DIG, LEDCtrl
    );
    input clk;
    input rst;
    input IOWrite;
    input LEDCtrl;
    input[1:0] LED_addr;
    input[15:0] writeData;

    output[7:0] Y;
    output[7:0] DIG;
```

测试说明

测试场景一

基本测试场景1

说明：使用Minisys开发板上的16+3个拨码开关用于做输入，其中3个拨码开关（sw23，sw22，sw21）用于测试用例的编号输入，16个拨码开关（sw15,..sw0）用于做测试数据的输入，使用17个led灯做输出

场景1.用例编号	用例描述
3'b000	输入测试数a，输入完毕后在led灯上显示a，同时用1个led灯显示a是否为二进制回文的判定（根据a的实际位宽来做判断，比如4'b1001是回文，该led灯亮，否则该led灯不亮）
3'b001	输入测试数a，输入完毕后在输出设备上显示a；输入测试数b；输入完毕后在输出设备上显示b
3'b010	计算 a & b，将结果显示在输出设备
3'b011	计算 a | b，将结果显示在输出设备
3'b100	计算 a ^ b，将结果显示在输出设备
3'b101	计算 a 逻辑左移 b位，将结果显示在输出设备
3'b110	计算 a 逻辑右移 b位，将结果显示在输出设备（测试时会考虑先做左移后再右移）
3'b111	计算 a 算数右移 b位，将结果显示在输出设备（测试时会考虑先做左移后再右移）

用例编号	输入/描述	结果
000	0000 1001 0110	sw17亮灯 sw17亮灯 sw17不亮灯
001	a=1100_1001_1000_0011|b=0000_0000_0000_0010	
010	a & b	0000_0000_0000_0010
011	a |b	1100_1001_1000_0011
100	a ^ b	1100_1001_1000_0001
101	a 逻辑左移 b位	0010_0110_0000_1100
110	a 逻辑右移 b位	0011_0010_0110_0000
111	a 算数右移 b位	1111_0010_0110_0000

测试场景二

基本测试场景2： 1/2

- 使用Minisys开发板上的3+8个拨码开关用于做输入，其中3个拨码开关（sw23，sw22，sw21）用于测试用例的编号输入，8个拨码开关（sw7,..sw0）用于做测试数据的输入（sw7对应于8bit的最高bit位bit7，sw0对应于8bit的最低bit位bit0）；在数据存储区域中申请4块连续区域用于分别存放下表中的4块数据，4块区域的大小一致（对应与下标中的space）；

数据集编号	数据集0	数据集1	数据集2	数据集3
数据集的起始地址	base + 0	base + 1*space	base + 2*space	base + 3*space
数据说明	输入数据原样保存	将输入数据视为无符号数，按从小到大排序后顺序存放的数据	将输入数据视为按照下标中的3'b010对应行的方式做识别，转换成补码后进行存放，存放次序与数据集0中的输入数据一致	基于"数据集2"中的数据，视其为有符号数的补码，按从小到大排序后顺序存放的数据

场景2.用例编号	用例描述
3'b000	输入测试数据的个数（小于或等于10），以2进制的方式输入多个测试数据，将测试数据原样存入上表 "数据集0" 对应的空间
3'b001	将测试数据视为无符号数（bit7与bit6~bit0一样，都被视为数值位的一部分），将数据按照从小到大的方式进行排序，排序后的数据集合作为一个整体存入上表 "数据集1" 对应的空间中（数值最小的存放在低地址，依次类推，数值最大的存放在高地址）
3'b010	将测试数据（此时"数据集0"中的每一个数据，其bit6~bit0被视为该数值的绝对值，bit7对应该数值的符号位）转换为有符号数的补码形式，将做完转换后的数据集存入上表 "数据集2" 对应的空间中（存放次序与数据集0中的数据一致）
3'b011	基于"数据集2"中的数据，视其为有符号数的补码，按从小到大排序后顺序存放的数据，排序后的数据集合作为一个整体存入上表 "数据集3" 对应的空间中（数值最小的存放在低地址，依次类推，数值最大的存放在高地址）

基本测试场景2:2/2

使用Minisys开发板上3（sw23，sw22，sw21用于测试用例编号的输入）+4（用于指定数据集内元素的下标）个拨码开关用于做输入，按照表格中指定的方式进行操作，输出指定结果

数据集编号	数据集0	数据集1	数据集2	数据集3
数据集的起始地址	base + 0	base + 1*space	base + 2*space	base + 3*space
数据说明	输入数据原样保存	将输入数据视为无符号数，按从小到大排序后顺序存放的数据	将输入数据视为按照下标中的3'b010对应行的方式做识别，转换成补码后进行存放，存放次序与数据集0中的输入数据一致	基于"数据集2"中的数据，视其为有符号数的补码，按从小到大排序后顺序存放的数据

场景2.用例编号	用例描述
3'b100	用数据集1中的最大值减去该数据集中的最小值，将结果显示在输出设备
3'b101	用数据集3中的最大值减去该数据集中的最小值，将结果显示在输出设备
3'b110	输入数据集编号（1或2或3）和该数据集中元素的下标(从0开始编址)，输出该数据的低8bit
3'b111	输入数据集中元素的下标，在输出设备上交替显示下面两组信息（间隔5秒）： 1）数据集0的编号（0）、用户指定的元素下标、数据集0中该下标对应的元素的低8bit 2）数据集2的编号（2）、用户指定的元素下标、数据集2中该下标对应的元素的低8bit

用例编号	输入/描述	结果
000	0000_0100（测试数的个数） 1000_1000 0001_0000 0000_0100 1000_0100	
001		数据集1的第0个数： 0000_1000 数据集1的第1个数： 0001_0000 数据集1的第2个数： 1000_0100 数据集1的第3个数： 1000_1000
010		数据集2的第0个数： 1111_1000 数据集2的第1个数： 0001_0000 数据集2的第2个数： 0000_1000 数据集2的第3个数： 1111_1100
011		数据集3的第0个数： 1111_1100 数据集3的第1个数： 1111_1000 数据集3的第2个数： 0000_1000 数据集3的第3个数： 0001_0000
100		0000_0000_1000_0000
101		1111_1111_0001_0100
110		查询上述数据集的数字，结果正确
111		进行交替闪烁，结果正确

问题及总结

问题

1.七段数码管显示频率CPU频率适配问题：

以Y18时钟作为分频让七段数码管显示会出现与CPU周期不同步的情况，从而错误读入了CPU内部流动的数据，导致显示数据不符合预期。经过对时钟周期的计算，解决了数码管稳定显示并且正确显示输入数据的难点。

2.MIPS指令逻辑问题：

通过lw和sw指令的正确使用，使得IO设备正确读入和写入数据。

通过对寄存器和指令的正确使用，实现各种循环，判断以及计算。

尽量避免在不同位置使用同一寄存器，从而保证数据准确性及稳定性。

3.CPU时钟问题：

通过矩阵键盘读入数据时，出现数据显示错误，输入响应过慢，读入脏数据等情况。通过重新整合CPU内部数据通路，时钟周期以及分频频率的关系，使得矩阵键盘得以正确显示以及读入数据。

4.DEGUG问题：

由于verilog以及mips文件中逻辑和执行方式不相同，在DEBUG过程中常常出现无从下手的情况。

通过控制代码变量的方式合理设计DEBUG环节，循序渐进地找出并解决问题。

总结

在设计及实现单周期CPU的过程中，我们深刻理解了CPU的数据通路与逻辑，加强了DEBUG能力，对汇编语言的使用也得到了锻炼，对硬件与软件的交互有了新的认识。

但是，还有许多的优化方向如下：

1.优化系统输入输出，避免组件中过多wire连接造成的接口连接错误、Multi-driver等问题。

2.优化结构化设计模式，将多个子模块分开设计，并在顶层设计中分开例化，用wire串联子模块。

3.在设计和实现过程中，提供丰富且清晰的代码注释，确保了小组合作的顺畅进行和代码的持续可用性。